Clustered Hierarchical Search Structure for Large-Scale Packet Classification on FPGA

Oğuzhan Erdem
Electrical and
Electronics Engineering
Middle East Technical University
Ankara, TURKEY 06800
Email: ogerdem@metu.edu.tr

Hoang Le
Ming Hsieh Department of
Electrical Engineering
University of Southern California
Los Angeles, USA 90007
Email: hoangle@usc.edu

Viktor K. Prasanna
Ming Hsieh Department of
Electrical Engineering
University of Southern California
Los Angeles, USA 90007
Email: prasanna@usc.edu

***Abstract*—Most current SRAM-based high-speed Internet Protocol (IP) packet classification implementations use tree traversal and pipelining. However, these approaches result in inefficient memory utilization. Due to the limited amount of on-chip memory of the state-of-the-art Field Programmable Gate Arrays (FPGAs), existing designs cannot support large filter databases arising in backbone routers and intrusion detection systems.**

Hierarchical search structures for packet classification exhibit good memory performance and support quick rule update. However, pipelined hardware implementation of these algorithms suffer from inefficient resource and memory usage due to variation in the size of the trie nodes and backtracking. We propose a memory efficient organization denoted Clustered Hierarchical Search Structure (*CHSS*) for packet classification. We present a clustering algorithm that partitions a given filter database to reduce the memory requirement. We show that, using the resulting structure, backtracking is not needed to perform a search. We introduce two parameters (NR_{trie}, NR_{tree}), which can be chosen based on the given filter database to achieve good memory efficiency. Our algorithm demonstrates substantial reduction in the memory footprint compared with the state-of-the-art. For all publicly available filter databases, the achieved memory efficiency is between 21.54 and 41.25 bytes per rule. We map the proposed data structure onto a linear pipeline architecture to achieve high throughput. Post place and route result using a state-of-the-art FPGA device shows that the design can sustain a throughput of 408 million packets per second, or 130.5 Gbps (for the minimum packet size of 40 Bytes).

I. Introduction

State-of-the-art FPGAs offer high operating frequency, unprecedented logic density and a host of other features. Additionally, FPGAs are programmed specifically for the problem to be solved. They can achieve higher performance than general-purpose processors. Thus, FPGA is a promising implementation technology for packet forwarding as well as for packet classification [3], [4], [9], [12], [14].

Most hardware-based solutions for high speed packet processing fall into two main categories: ternary content addressable memory (TCAM)-based and dynamic/static random access memory (DRAM/SRAM)-based solutions. Although TCAM-based engines can retrieve results in just one clock cycle, their throughput is limited by the relatively low speed of TCAMs. They are expensive and offer little adaptability to new addressing and routing protocols [1]. Since SRAM-based solutions utilize some kind of tree traversal, they require multiple cycles to process a packet. Several researchers have explored pipelining to improve the throughput. However, these designs suffer from inefficient memory utilization, which limits the number of supported filters, and/or poor support for incremental update. These have been the dominant issues for packet classification implementations on FPGA.

The key challenges to be addressed in designing an architecture for IP packet header classification are (1) size of supported ruleset, (2) high throughput, (3) scalability, and (4) incremental update. To address these issues, we propose and implement a scalable, high-throughput, and memory-efficient SRAM-based linear pipeline architecture for packet classification on FPGAs, namely *CHSS*. This paper makes the following contributions:

1) A clustering algorithm that partitions a given filter database to eliminate backtracking in the state-of-the-art hierarchical search structures (Section III).
2) A three-stage parameterized hierarchical search structure for packet classification. The parameters of the design can be tuned for a given ruleset to improve the memory efficiency (Section III).
3) A linear pipelined SRAM-based architecture that can be easily implemented in hardware (Section IV).
4) A design that achieves memory efficiency between 21.54 and 41.25 bytes per filter, and sustains a high throughput of 408 million packets per second on a state-of-the-art FPGA device (Section V).

The rest of the paper is organized as follows. Section II covers the background and related work. Section III details the proposed clustering algorithm and data structure. Section IV introduces the proposed architecture and its implementation on FPGA. Section V presents experimental setup and implementation results. Section VI concludes the paper.

II. Background

A. Notations

The following notations are used throughout the paper: *SA* - Source Address, *DA* - Destination Address, *PRTCL* - Protocol,

Supported by the U.S. National Science Foundation under grant No. CCF-1018801. Equipment grant from Xilinx is gratefully acknowledged.

SP - Source Port, and *DP* - Destination Port.

B. Packet classification overview

Packet classification is one of the fundamental challenges in designing high speed router. It enables the router to support firewall processing, quality of service differentiation, virtual private networks, policy routing, and other value added services. Therefore, packet header classification is an essential part of a full-featured network router. An IP packet is classified based on 5-tuple header filters (i.e. $\langle SA, DA, PRTCL, SP, DP \rangle$), in which fields are generally specified by prefixes or ranges. When a packet arrives at a router, its header is compared against a set of rules, often known as a ruleset or filter database. Each rule can have one or more fields and their associated value, a priority, and an action to be taken if matched. A packet is considered matching a rule only if it matches all the fields within that rule. A sample ruleset is shown in Table I. The terms *ruleset* and *filter database* are used interchangeably in this paper.

TABLE I
SAMPLE 5-FIELD RULESET

Rule	SA	DA	SP	DP	PRTCL	Priority	Action
R1	0*	10*	80	*	TCP	1	Act0
R2	0*	01*	17	17	UDP	2	Act1
R3	0*	1*	44	*	UDP	2	Act2
R4	00*	1*	17	44	UDP	3	Act3
R5	00*	11*	*	100	TCP	4	Act4
R6	10*	1*	*	*	*	5	Act5
R7	*	00*	*	*	TCP	5	Act6
R8	0*	10*	*	100	TCP	6	Act7
R9	0*	1*	*	*	TCP	7	Act8
R10	0*	10*	17	17	UDP	7	Act9
R11	111*	000*	80	*	TCP	8	Act10

C. Related Work

Many proposed packet header classification algorithms and architectures are based on decision trees, which take the geometric view of the packet classification problem. HiCuts [5] and its enhanced version HyperCuts [16] are representatives of such algorithms. The common problems of these approaches are the large variance in the memory efficiency due to rule duplication and the lack of support for incremental updates. Decision forest [9] proposed a partitioning algorithm to reduce the rule duplication during the construction of the decision trees. However, this work was aimed to solve a different problem in which the number of fields is extended from 5 to 11 fields. Thus, it is not clear how effective this approach is when applying to the classic 5-field packet classification.

Another popular approach is hierarchical-trie (H-trie). An H-trie is built using SA and DA prefixes. Initially, a SA trie is constructed. Each prefix node of the SA trie hierarchically connects a DA trie. A prefix node in a DA trie stores rules which have the same source prefix field. Search operation starts from the SA trie. If a match occurs then the corresponding DA trie is traversed. Although a match can be found in any node of the DA trie, search has to backtrack to the SA trie and repeats the same operation for the next matched SA prefix to find the highest priority match. The search terminates when all the matched nodes in the SA trie are visited. Set-pruning trie eliminates the backtracking by replicating the rules [20]. Grid-of-tries (GoT) [18] data structure for 2-field packet classification avoids the backtracking by introducing switch pointers to some trie nodes; and hence each rule is stored in only one node. In GoT, all the matching rules are not traversed because a rule with a longer destination prefix length is assumed to have a higher priority than a rule with a shorter destination prefix length. The authors in [2] presented the Extended Grid-of-tries (EGT) to improve the previous idea to support 5-field packet classification. Each node in EGT has a pointer to a list of rules. Whenever a matching node is reached, linear search is performed in the corresponding rule list. Although EGT has good memory performance, large number of worst-case memory accesses decreases the search time performance.

III. ALGORITHM AND DATA STRUCTURE

A. Definitions

Definition ***Prefix node*** in a trie is any node to which a path from the root of the trie corresponds to a SA or DA prefix of a rule. If a prefix node is a leaf node then it is called ***leaf prefix node***, otherwise ***non-leaf prefix node***. If there is no valid prefix stored in a trie node, then it is called a ***non-prefix node***.

Definition Two prefixes *x*, *y* are said to be ***disjoint*** if *x* is not a prefix of *y* and *y* is not a prefix of *x*. A prefix set such that all pairs of prefixes are disjoint is called a ***disjoint prefix set***.

Definition Two rules are said to be ***overlapped*** if they have the same SA and DA prefixes. For instance, R1, R8 and R10 in Table I are overlapped.

Definition ***Memory efficiency*** is defined as the average amount of memory (in bytes) required to store a filter rule.

B. Clustering Algorithm

While hierarchical trie structures can easily be implemented on multi-core network processors, the hardware implementation of these algorithms has two issues: (1) backtracking requirement and (2) memory inefficiency. There are 3 types of **backtracking**: (1) from a DA trie node to a SA trie node, (2) from a DA trie node to another DA trie node within the same DA trie, and (3) from a DA trie node to another DA trie node of a different DA trie. In these cases, the search needs to proceed in the backward direction and requires stalling the pipeline. The **memory inefficiency** is due to the variable number of rules stored in each node of the search structure. This number ranges from 0 to 73 for different rulesets [19]. In hardware implementation, the size of a node is determined by that of the largest node, leading to excessive and inefficient memory usage. Memory efficiency issue is addressed in Section III-C.

We propose a clustering algorithm to partition a given ruleset based on the SA field to eliminate backtracking from

a DA trie to the SA trie. The remaining types of backtracking are addressed in Section III-C. The algorithm takes a set of prefixes as its input and generates a collection of non-empty subsets $\{S_i\}$. Each subset is called a *cluster*. In each cluster, the prefixes are pairwise disjoint; and the trie representation of S_i, or simply S_i trie, only contains prefixes at the leaf nodes. Each S_i trie has its own set of DA tries. Each trie corresponds to a leaf node of the S_i trie. Note that in the worst case, the number of clusters is 32 for IPv4. However, our analysis of the real life and synthetic rulesets collected from [19] shows that the number of clusters is at most 4. The pseudo-code is given in Algorithm 1.

Algorithm 1 Clustering algorithm

Input: Prefix set S
Output: A partition of S into a collection of non-empty prefix subsets such that within each subset all the prefixes are pairwise disjoint

```
num_sets = 0
Sort the prefixes in S by length
while S ≠ φ do
    S_num_sets ← φ, S' ← φ
    while S ≠ φ do
        Let P_j be the longest prefix in S
        if P_j is disjoint with all the prefixes in S_num_sets then
            Add P_j to S_num_sets
        else
            Add P_j to S'
        end if
        Delete P_j from S
    end while
    S = S', num_sets = num_sets + 1
end while
return {S_i}, 0 ≤ i < num_sets
```

C. Clustered Hierarchical Search Structure (CHSS)

The proposed hierarchical data structure consists of 3 stages:
Stage 1 **(SA trie)**: One binary trie is built for each cluster using the SA prefixes. These SA tries have 2 properties: (1) all the prefixes are at the leaf nodes and (2) no rules are stored at any node.
Stage 2 **(DA trie)**: Each leaf node of the SA tries connects to a DA trie. Therefore, the number of DA tries equals to the number of SA prefixes within a cluster. Each prefix node of a DA trie has at least one rule. For each rule, only the SP, DP, PRTCL, and Priority fields are stored. The prefix nodes of the DA trie can be a *leaf prefix node* or a *non-leaf prefix node*. Also, in this stage, there is no sharing of DA tries to eliminate the second case of backtracking.
Stage 3 **(DP tree)**: A range tree is built for each *leaf prefix node* of a DA trie if the number of rules at the node exceeds a predefined threshold. The range tree is built using the DP number of the rules. Fig. 1 illustrates a sample DP range table and its corresponding range tree. The search space is divided into *disjoint* range intervals. Each interval corresponds to a single node of the range tree. Each node in the range tree has at least one rule. For each rule, only the SP, PRTCL, and Priority fields are stored.

Note that, in Stage 3, SP trees can also be used instead of the DP trees. However, our analysis of the real life and synthetic rulesets shows that the rules are distributed more evenly between the nodes in the DP trees than in the SP trees. Also, in Stage 3, the number of rules per node is relatively small; hence, additional stages are not needed.

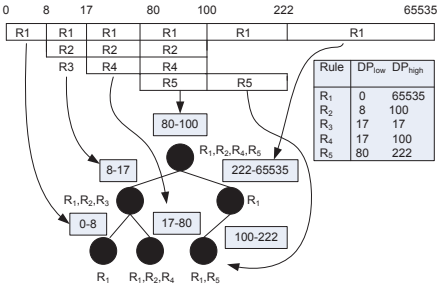


Fig. 1. Range tree data structure (DP-tree)

We define two design parameters:
NR$_{\text{trie}}$: Our data structure stores the overlapped rules in the DA trie nodes rather than pointing to a list of these rules. However, the number of rules in each node is not constant. This results in memory inefficiency in hardware implementation. To improve the memory efficiency, we set a limit on the number of overlapped rules per trie node, NR_{trie}. The remaining overlapped rules are either used to construct a DP tree (if they belong to a leaf node of DA trie), or otherwise moved to a TCAM (the *non-leaf trie rules*).
NR$_{\text{tree}}$: In the DP tree, the number of rules per node can vary. We also set a limit on the number of rules per tree node, NR_{tree}. The remaining rules of the DP trees are also moved to TCAM (the *excessive tree rules*).

Note that the rules moved to TCAM include: (1) the non-leaf trie rules and (2) the excessive tree rules. For each rule, all the 6 fields are stored, i.e. $\langle SA, DA, PRTCL, SP, DP, Priority\rangle$. Fig. 2 illustrates the *CHSS* using the sample ruleset shown in Table I (with $NR_{trie} = NR_{tree} = 1$). In general, NR_{trie} and NR_{tree} provide the trade-off between the size of the on-chip memory and TCAM. These parameters also affect the memory efficiency. Larger values of NR_{trie} and NR_{tree} decrease the number of rules stored in TCAM; and hence the size of TCAM. However, this decreases the memory efficiency. Additionally, we use path compression [13] for trie structures to further improve the memory efficiency. Path compression performs particularly well with the sparse trie structures, making *CHSS* well suited for IPv6.

D. Design Methodology

Several design problems can be formulated to choose the design parameters. Due to space limitation, we briefly describe one design methodology. Let α denote the ratio of the number of rules stored in the TCAM over the total number of rules of the given ruleset. An example design problem is as follows. Given a ruleset and an $\alpha_T \geq 0$ as the input threshold, choose NR_{trie} and NR_{tree} such that $\alpha \leq \alpha_T$. Let NR_{max} be the maximum number of rules that can be stored at each node over the entire structure. First, NR_{max} can be easily determined

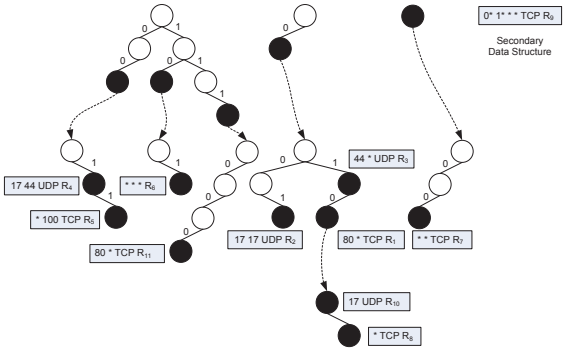


Fig. 2. The *CHSS* data structure for the sample ruleset in Table I

by building the *CHSS* for the given ruleset with $\alpha = 0$ and finding the maximum number of rules per node over the entire *CHSS*. Secondly, we independently vary NR_{trie} and NR_{tree} from 1 to NR_{max}. For each $\langle NR_{trie}, NR_{tree} \rangle$ pair, a *CHSS* is generated and the resulting memory efficiency and α are calculated. A design such that $\alpha \leq \alpha_T$ is selected. In the case that more than one design satisfies the constraint, the design with the highest memory efficiency (or lowest memory requirement) is returned.

E. Packet classification algorithm

For each incoming packet, all the 5 fields (SA, DA, $PRTCL$, SP, DP) are extracted from the header and forwarded to all the *CHSS* clusters. Search is performed in all the clusters in parallel. The search operation in each stage is carried out as follows:

SA trie: Search starts from the root node and continues traversing the SA trie using the bits of the source IP address. The traversal is determined by the most significant bit of the SA (left if 0 or right otherwise). Search operation in the SA trie terminates when a leaf node or a null pointer is reached. If the search terminates in a leaf node, then the root address of the corresponding DA trie is obtained.

DA trie: Search uses the destination IP address to traverse the DA trie. In each node, the source port number *SP*, the destination port number *DP* and the protocol *PRTCL* of the packet header are compared with the corresponding fields of the rules stored in that node in parallel. If the priority value of the matched rule is higher than that of the last match, then the search result is updated. As in the SA trie, the search operation terminates when a leaf node or a null pointer is reached. If search ends at a leaf node which points to a root of a DP tree, then the root address of the corresponding DP tree is obtained.

DP tree: The destination port number of the incoming packet is used to traverse the DP tree. At each node, the *SP* and *PRTCL* fields of the packet are compared with the corresponding fields of the rules stored in that node in parallel. Again, if the priority value of the matched rule is higher than that of the last match, then the match result is updated.

TCAM: Search in TCAM is performed in a single cycle. All the fields of the packet header are compared with all the TCAM entries in parallel.

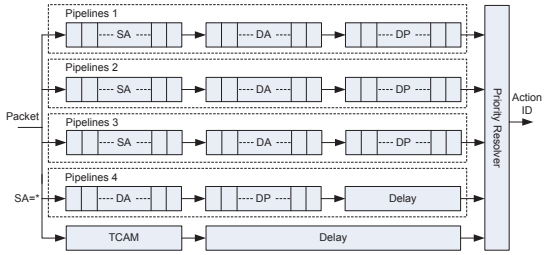


Fig. 3. Multi-pipeline *CHSS* architecture

The search results from all the clusters are compared based on their priority value. The matching rule with the highest priority is returned as the final result.

Claim: Three types of *backtracking* are not needed in the search using *CHSS*.

Proof: Type 1: The proposed clustering algorithm ensures that all the SA prefixes within any cluster are disjoint. Thus, at most one match is possible in each SA trie. Thus, once the search leaves Stage 1 (SA trie), it does not have to jump back. **Type** 2: In Stage 2, matching results can be resolved at each node as rules are stored locally at the node. Hence, the search can simply move in the forward direction. **Type** 3: There is no sharing of DA tries between the S_i tries. Therefore, the search does not have to jump to other DA trie nodes and backtracking is not required. ■

IV. Architecture and Implementation on FPGA

A. Architecture

Fig. 3 describes the overall architecture of the proposed *CHSS* engine for packet classification. Pipelining is used to improve the throughput. SA and DA denote pipelines for the source and destination address prefix tries, respectively. DP represents the pipeline for the DP trees. Each cluster has 3 pipelines: SA, DA, and DP. The collection of DA tries associated with the leaf nodes of the SA trie within the same cluster are mapped level by level onto the same DA pipeline. Similarly, the collection of DP trees associated with the leaf nodes of the DA trie within the same cluster are mapped level by level onto the same DP pipeline. Therefore, the total number of pipelines equals to the number of clusters $\times 3$. In the worst-case, there are 32×3 pipelines for IPv4. However, the worst-case scenario does not occur in practice. The number of stages in each pipeline is determined by the height of the tree structure. The SA and DA pipelines have 32 stages in the worst case for IPv4. The depth of DP pipeline depends on the number of disjoint DP ranges in the rule. Delay blocks are added at the end of the shorter DP tree paths and the TCAM to match the latency of the pipelines.

The BRAM of the state-of-the-art FPGAs supports dual-ported feature. To take advantage of it, the architecture is configured as dual linear pipelines to double the search rate. These two pipelines have their own logic, but share the same memory in each stage. The memory has dual Read/Write ports so that two packets can be input every clock cycle.

TABLE II
MEMORY EFFICIENCY (BYTES PER RULE) FOR VARIOUS RULESETS

1	2	3	4	5	6	7	8	9	10	11
Ruleset	N	Set 1	Set 2	Set 3	Set 4	*CHSS*	EGT [2]	HyperCuts [16]	BV [11]	Hybrid scheme [8]
ACL	752	679	16	55	2	25.11	25.41	32.58	71.80	N/A
ACL100	98	85	3	10	0	21.54	27.69	27.78	47.35	24.44
ACL1K	916	798	28	85	5	22.44	24.96	38.15	91.63	22.98
ACL5K	4415	3927	279	201	8	23.95	24.87	59.64	257.23	24.83
ACL10K	9603	8416	496	684	7	22.21	30.23	54.22	789.22	25.51
FW	269	93	6	1	169	21.76	25.31	399.18	40.72	N/A
FW100	92	28	2	2	60	27.53	23.42	113.37	27.46	56.63
FW1K	791	268	23	12	488	23.84	23.80	6110.58	67.08	215.06
FW5K	4653	1554	264	34	2801	35.37	39.04	16132.65	691.69	255.13
FW10K	9311	3611	28	0	5672	41.25	49.45	12554.18	1582.18	248.54
IPC	1550	1207	201	13	129	21.80	26.63	128.52	61.57	N/A
IPC100	99	65	18	3	13	23.65	31.60	24.57	69.16	23.65
IPC1K	938	730	101	47	60	22.22	29.95	61.34	176.03	25.63
IPC5K	4460	3763	218	147	332	21.72	27.62	406.80	358.61	49.46
IPC10K	9037	7659	491	331	556	22.60	28.92	2378.35	788.69	43.30

B. Implementation

As previously stated, the two parameters NR_{trie} and NR_{tree} can be chosen based on a given ruleset to use the available BRAM efficiently. In our design, we set a limit on $\alpha_T = 0.1$ (the ratio of the number of rules stored in the TCAM over the total number of rules of the given ruleset). This ratio can be relaxed utilizing larger TCAMs to further improve the memory efficiency. Given α_T, NR_{trie} and NR_{tree} can be determined using the design methodology shown in Section III-D. Furthermore, each pipeline of the architecture can be independently implemented using different set of parameters to achieve better memory efficiency.

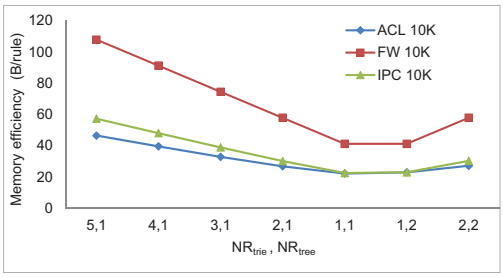
V. PERFORMANCE EVALUATION

A. Memory requirement

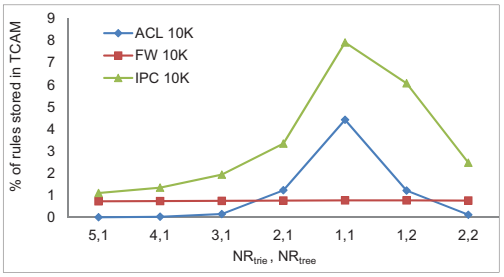
Fifteen publicly available rulesets were collected from [19]. There are three different types of rulesets: Access Control List (ACL), Firewall (FW), and IP Chain (IPC). Each group has 5 different filter sets, whose sizes range from 100 to 10K entries.

We applied the proposed clustering algorithm to each ruleset. Our experimental results show that each ruleset has at most 4 clusters. The total number of rules in each ruleset (Column 2) and the size of each cluster are shown in Table II (Columns 3-6). Note that Set 4 includes rules whose SA length is zero (default prefix SA=*). Also in Table II, Column 7 shows the best memory efficiency for each ruleset using the corresponding $\langle NR_{trie}, NR_{tree} \rangle$ pair. The memory efficiency is computed by dividing the total amount of memory required to store the entire *CHSS* by the total number of rules (excluding the rules stored in TCAM). Each node in the entire data structure is assumed to have 2 child pointers, requiring 2 bytes each. Columns 8-11 show the results of the existing approaches for the same rulesets.

NR_{trie} and NR_{tree} parameters provide the trade-off between the amount of memory used in the pipelines of *CHSS* and TCAM. Our experiments using a collection of real life tables show that, when NR_{trie} and NR_{tree} are increased, the size of the main data structure increases, and the size of TCAM reduces. Furthermore, smaller values for these parameters provide better memory efficiency; however, the size of TCAM increases. Fig. 4(a) shows the memory efficiency of the largest rulesets (ACL10K, FW10K and IPC10K) for various NR_{trie} and NR_{tree} values. Fig. 4(b) demonstrates the impact of NR_{trie} and NR_{tree} on the percentage of the total number of rules stored in TCAM over total number of rules for the largest ruleset in each case.



(a) Memory efficiency



(b) Percentage of the number of rules stored in TCAM

Fig. 4. Memory efficiency (Bytes/rule) and percentage of rules stored in TCAM for 3 largest rulesets

B. Throughput

The proposed hardware design was implemented in Verilog, using Xilinx ISE 12.4, with Xilinx Virtex-5 XC5VFX200T with -2 speed grade as the target. The architecture supports the largest ruleset ACL10K consisting of 9603 rules. The post

TABLE III
PERFORMANCE COMPARISON

1	2	3	4	5	6
Packet classification engines	Platform	# of rules	Memory efficiency (Bytes/rule)	Throughput (Gbps)	Throughput efficiency (Gbps/B)
CHSS	FPGA	9603	22.21	130.5	5.88
Hybrid scheme [8]	FPGA	9603	25.51	80.00	3.14
Optimized HyperCuts [7]	FPGA	9603	63.73	80.23	1.26
Simplified HyperCuts [10]	FPGA	10000	28.60	10.84	0.38
BV-TCAM [17]	FPGA	222	72.07	10.00	0.14
2sBFCE [14]	FPGA	4000	44.50	2.06	0.05
Memory-based DCFL [6]	FPGA	128	1726.56	24.00	0.01
B2PC [15]	ASIC	3300	163.63	13.60	0.08

TABLE IV
IMPLEMENTATION RESULTS

Clock period (ns)	Frequency (MHz)	Number of Slices	BRAM (36-Kb block)
4.894	204	23064	55

place and route results are collected in Table IV. Using dual-ported memory, the design can support 408 million packets per second (MPPS), or 130.5 Gbps for the minimum packet size of 40 bytes (or 320 bits). The designs that support other experimental rulesets are much simpler; hence, they can achieve a higher throughput.

C. Performance Comparison

The performance of *CHSS* is compared with the state-of-the-art packet classification approaches with respect to the memory efficiency in Bytes/rule, throughput in Gbps, and throughput efficiency in Gbps/Bytes (the ratio of the throughput to the memory efficiency). The results for the existing designs were reported in [8]. Note that all the designs have been implemented on a Xilinx Virtex-5 device for fair comparisons. Columns 7-11 in Table II show that, our scheme exhibits the superior memory efficiency, compared with the existing approaches for the same rulesets. Furthermore, the variation of the memory efficiency in *CHSS* is smaller than that of the other solutions. Table III gives the throughput and throughput efficiency of the state-of-the-art hardware-based packet classification engines. Column 5 shows that our design achieves the highest throughput performance among all the existing architectures. Our scheme also outperforms all the existing schemes with respect to the throughput efficiency, as shown in Column 6.

VI. CONCLUSION

In this paper, we proposed *CHSS*, a memory efficient clustered hierarchical data structure for packet classification. Our experimental results show that *CHSS* outperforms the state-of-the-art approaches with respect to memory efficiency. A drawback of our design is the need of TCAM. Additionally, the number of clusters generated by our clustering algorithm is ruleset-dependent. In the future work, we plan to (1) minimize or eliminate the TCAM, (2) modify the proposed algorithm to output a fixed number of clusters, (3) extend the idea to support the IPv6 standard, and (4) support packet classification in virtual routers.

REFERENCES

[1] F. Baboescu, S. Rajgopal, L. Huang, and N. Richardson. Hardware implementation of a tree based IP lookup algorithm for oc-768 and beyond. In *Proc. DesignCon '05*, 2005.

[2] F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: Is there an alternative to cams. In *In IEEE INFOCOM*, 2003.

[3] G. Brebner. Reconfigurable computing for high performance networking applications. In A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, and T. El-Ghazawi, editors, *Reconfigurable Computing: Architectures, Tools and Applications*, volume 6578 of *Lecture Notes in Computer Science*, pages 1–1. Springer Berlin / Heidelberg, 2011.

[4] J. Dharmapurikar, S. Lockwood. Fast and scalable pattern matching for network intrusion detection systems. *IEEE Journal on Selected Areas in Communications*, 24(10):1781–1792, 2006.

[5] P. Gupta and N. Mckeown. Packet classification using hierarchical intelligent cuttings. In *in Hot Interconnects VII*, pages 34–41, 1999.

[6] G. S. Jedhe, A. Ramamoorthy, and K. Varghese. A scalable high throughput firewall in fpga. In *Proceedings FCCM*, pages 43–52, Washington, DC, USA, 2008. IEEE Computer Society.

[7] W. Jiang and V. K. Prasanna. Large-scale wire-speed packet classification on fpgas. In *Proceedings of the FPGA 2009*, FPGA '09, pages 219–228, New York, NY, USA, 2009. ACM.

[8] W. Jiang and V. K. Prasanna. Scalable packet classification: Cutting or merging? In *Proceedings of the ICCCN '09*, ICCCN '09, pages 1–6, Washington, DC, USA, 2009. IEEE Computer Society.

[9] W. Jiang, V. K. Prasanna, and N. Yamagaki. Decision forest: A scalable architecture for flexible flow matching on fpga. *International Conference on Field Programmable Logic and Applications*, 0:394–399, 2010.

[10] A. Kennedy, X. Wang, Z. Liu, and B. Liu. Low power architecture for high speed packet classification. In *ANCS'08*, pages 131–140, 2008.

[11] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. *SIGCOMM Comput. Commun. Rev.*, 28:203–214, October 1998.

[12] H. Le and V. K. Prasanna. Scalable high throughput and power efficient ip-lookup on fpga. In *Proc. FCCM '09*, 2009.

[13] D. R. Morrison. Patriciapractical algorithm to retrieve information coded in alphanumeric. *Journal ACM*, 15:514–534, October 1968.

[14] A. Nikitakis and L. Papaefstathiou. A memory-efficient fpga-based classification engine. *Proceedings FCCM*, 0:53–62, 2008.

[15] I. Papaefstathiou and V. Papaefstathiou. Memory-efficient 5d packet classification at 40 gbps. In *Proceedings INFOCOM*, pages 1370 –1378, May 2007.

[16] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *Proc. SIGCOMM*, SIGCOMM '03, pages 213–224, New York, NY, USA, 2003. ACM.

[17] H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *Proc. FPGA*, FPGA '05, pages 238–245, New York, NY, USA, 2005. ACM.

[18] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. *SIGCOMM Comput. Commun. Rev.*, 28:191–202, October 1998.

[19] D. E. Taylor and J. S. Turner. Classbench: a packet classification benchmark. *IEEE/ACM Trans. Netw.*, 15:499–511, June 2007.

[20] P. Tsuchiya. A search algorithm for table entries with non-contiguous wildcarding. Unpublished report. Bellcore.